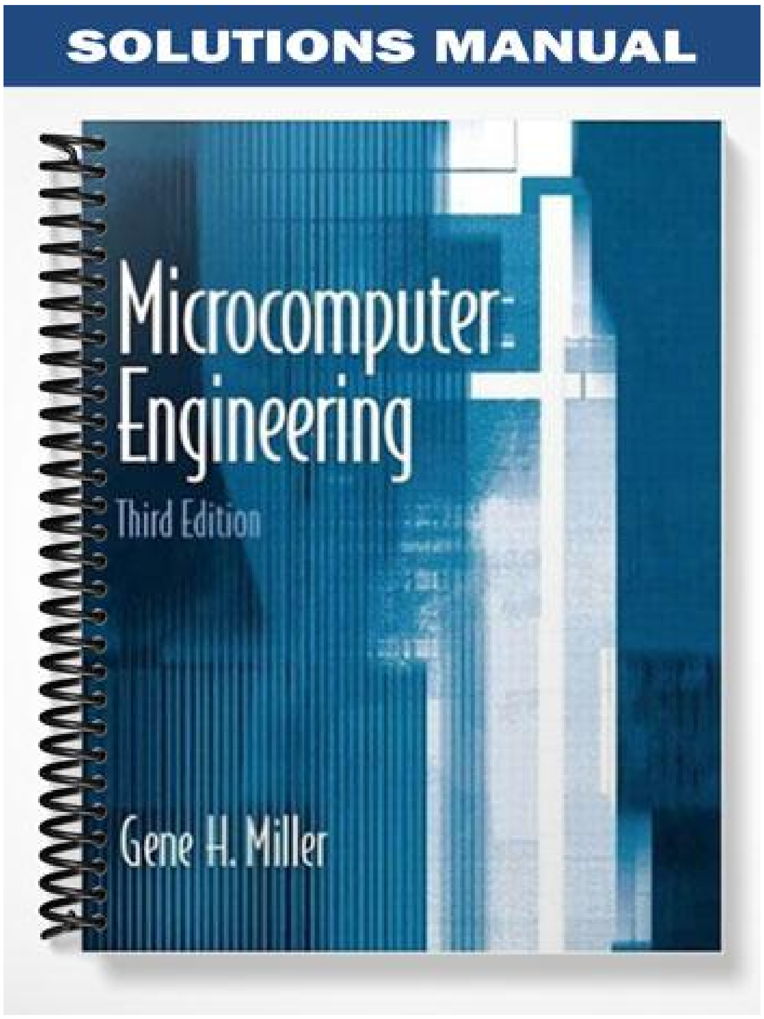

SOLUTIONS MANUAL

Microcomputer: Engineering

Third Edition

Gene H. Miller

PREDICTIVE CONTROL
WITH CONSTRAINTS
SOLUTIONS MANUAL

J.M.Maciejowski
Cambridge University Engineering Department

3 December 2001

World-Wide Web site for this book: `http://www.booksites.net/maciejowski/`

Contents

Chapter 1

Introduction

1.1 Assuming a perfect model and no disturbances, $y(k+1) = 2.2864$. Hence using $\hat{y}(k+2|k+1) = 0.7y(k+1) + 2u(k+1)$, the free response is given by assuming that $u(k+1) = u(k+2) = 0.4432$:

$$\hat{y}_f(k+2|k+1) = 0.7 \times 2.2864 + 2 \times 0.4432 = 2.4869 \tag{1}$$
$$\hat{y}_f(k+3|k+1) = 0.7 \times 2.4869 + 2 \times 0.4432 = 2.6272 \tag{2}$$

We have $\epsilon(k+1) = s(k+1) - y(k+1) = 3 - 2.2864 = 0.7136$, hence

$$r(k+3|k+1) = s(k+3) - \lambda^2 \epsilon(k+1) = 3 - 0.7165^2 \times 0.7136 = 2.6337 \tag{3}$$

Hence

$$\begin{aligned} \Delta\hat{u}(k+1|k+1) &= \frac{2.6337 - 2.6272}{3.4} = 0.0019 && (4) \\ u(k+1) = \hat{u}(k+1|k+1) &= u(k) + \Delta\hat{u}(k+1|k+1) && (5) \\ &= 0.4432 + 0.0019 = 0.4451 && (6) \end{aligned}$$

That is one step done.

Now the second step:

$$y(k+2) = 0.7 \times 2.2864 + 2 \times 0.4451 = 2.4907 \tag{7}$$
$$\hat{y}_f(k+3|k+2) = 0.7 \times 2.4907 + 2 \times 0.4451 = 2.6337 \tag{8}$$
$$\hat{y}_f(k+4|k+2) = 0.7 \times 2.6337 + 2 \times 0.4451 = 2.7338 \tag{9}$$
$$\epsilon(k+2) = s(k+2) - y(k+2) = 3 - 2.4907 = 0.5093 \tag{10}$$
$$r(k+4|k+2) = 3 - 0.7165^2 \times 0.5093 = 2.7385 \tag{11}$$
$$\Delta\hat{u}(k+2|k+2) = \frac{2.7385 - 2.7338}{3.4} = 0.0014 \tag{12}$$
$$u(k+2) = 0.4451 + 0.0014 = 0.4469 \tag{13}$$

Verification: $y(k+2) = 2.4907 \neq 2.2864 = 0.7 \times 2.0 + 2 \times 0.4432 = \hat{y}(k+2|k)$.

1.2 Here is the solution as a *MATLAB* file:

```
% Solution to Exercise 1.2:
setpoint = 3;
Tref = 9;  % Reference trajectory time constant
Ts = 3;    % sample time
lambda = exp(-Ts/Tref);
P = [1;2]; % coincidence points

% Initial conditions:
yk = 2;      % output y(k), from Example 1.3 (note y(k)=y(k-1)).
uk = 0.4429; % last applied input signal u(k) (from Example 1.4)
ykplus1 = 0.7*yk + 2*uk; % output y(k+1)

% Step response vector [S(P(1));S(P(2))] as in (1.21):
S = [2.0 ; 3.4]; % from Example 1.4

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Next step:
% Reference trajectory:
error = setpoint-ykplus1;
% Form reference (or target) vector [r(k+2|k+1) ; r(k+3|k+1)]:
T = [setpoint;setpoint]-[lambda^P(1);lambda^P(2)]*error;

% Free response vector:
yf1 = 0.7*ykplus1 + 2*uk;  % yf(k+2|k+1)
yf2 = 0.7*yf1 + 2*uk;      % yf(k+3|k+1)
Yf = [yf1 ; yf2]; % vector of free responses - as in (1.21)

% New optimal control signal:
Deltau = S\(T-Yf);  % as in (1.22)
ukplus1 = uk + Deltau(1) % use first element only of Deltau

% Resulting output:
ykplus2 = 0.7*ykplus1 + 2*ukplus1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% One more step:
% Reference trajectory:
error = setpoint-ykplus2;
% Form reference (or target) vector [r(k+3|k+2) ; r(k+4|k+2)]:
T = [setpoint;setpoint]-[lambda^P(1);lambda^P(2)]*error;

% Free response vector:
yf1 = 0.7*ykplus2 + 2*ukplus1;  % yf(k+3|k+2)
yf2 = 0.7*yf1 + 2*ukplus1;      % yf(k+4|k+2)
```

```
Yf = [yf1 ; yf2]; % vector of free responses - as in (1.21)

% New optimal control signal:
Deltau = S\(T-Yf);  % as in (1.22)
ukplus2 = ukplus1 + Deltau(1) % use first element only of Deltau

% Resulting output:
ykplus3 = 0.7*ykplus2 + 2*ukplus2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This gives the following results:

$$\begin{array}{ll} u(k+1) = 0.4448 & y(k+2) = 2.4897 \\ u(k+2) = 0.4463 & y(k+3) = 2.6354 \end{array}$$

In this *MATLAB* program the computations for each step have been set out separately. In practice one would set them out once inside a repeating loop — see program `basicmpc`.

1.3 **Repeat of Example 1.3:**
As in Example 1.3 we have $\epsilon(k) = 1$. Hence

$$\epsilon(k+2) = \left(1 - \frac{2T_s}{T_{ref}}\right)\epsilon(k) = \frac{1}{3} \tag{14}$$

Hence $r(k+2|k) = 3 - \frac{1}{3} = 2.6667$. So proceeding as in Example 1.3,

$$\begin{aligned} \Delta\hat{u}(k|k) &= \frac{2.6667 - 2.0}{3.4} = 0.1961 \qquad (15) \\ \hat{u}(k|k) &= u(k-1) + \Delta\hat{u}(k|k) = 0.4961 \qquad (16) \end{aligned}$$

This should result in the next plant output value being $y(k+1) = 0.7 \times 2 + 2 \times 0.4961 = 2.3922$.

Repeat of Example 1.4:
The vector of reference trajectory values (or 'target' vector) at the coincidence points is now

$$\mathcal{T} = \begin{bmatrix} 3 - \frac{2}{3} \\ 3 - \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 2.3333 \\ 2.6667 \end{bmatrix} \tag{17}$$

while the vectors $\mathcal{Y}_f$ and $\mathcal{S}$ remain unchanged. Hence

$$\begin{aligned} \Delta\hat{u}(k|k) &= \mathcal{S}\backslash(\mathcal{T} - \mathcal{Y}_f) = 0.1885 \qquad (18) \\ \hat{u}(k|k) &= u(k-1) + \Delta\hat{u}(k|k) = 0.4885 \qquad (19) \end{aligned}$$

This should result in the next plant output value being $y(k+1) = 0.7 \times 2 + 2 \times 0.4885 = 2.3770$.

1.4 *There is an error in equation (1.23) in the book.* Since the free response $\hat{y}_f(k+P_i|k)$ is defined to be the response when the input remains at its last value, namely $u(k-1)$, each term in (1.23) involving an input value $\hat{u}(k+j|k)$ should in fact involve the difference $\hat{u}(k+j|k)-u(k-1)$. Thus the correct expression for (1.23) is:

$$\begin{aligned}\hat{y}(k+P_i|k) = \hat{y}_f(k+P_i|k) + H(P_i)[\hat{u}(k|k)-u(k-1)] &+ \\ H(P_i-1)[\hat{u}(k+1|k)-u(k-1)] &+ \cdots \\ H(P_i-H_u+2)[\hat{u}(k+H_u-2|k)-u(k-1)] &+ \\ S(P_i-H_u+1)[\hat{u}(k+H_u-1|k)-u(k-1)]&\end{aligned} \tag{20}$$

This can be written as

$$\begin{aligned}\hat{y}(k+P_i|k) = \hat{y}_f(k+P_i|k) \ + [S(P_i)-S(P_i-1)][\hat{u}(k|k)-u(k-1)]+ \\ [S(P_i-1)-S(P_i-2)]\hat{u}(k+1|k)+\cdots \\ [S(P_i-H_u+2)-S(P_i-H_u+1)]\hat{u}(k+H_u-2|k)+S(P_i-H_u+1)\hat{u}(k+H_u-1|k)\end{aligned} \tag{21}$$

The terms can be regrouped as:

$$\begin{aligned}\hat{y}(k+P_i|k) &= \hat{y}_f(k+P_i|k)+S(P_i)[\hat{u}(k|k)-u(k-1)]+ \\ &\quad S(P_i-1)[\hat{u}(k+1|k)-\hat{u}(k|k)]+\cdots \\ &\quad S(P_i-H_u+1)[\hat{u}(k+H_u-1|k)-\hat{u}(k+H_u-2|k)] \\ &= \hat{y}_f(k+P_i|k)+S(P_i)\Delta\hat{u}(k|k)+ \\ &\quad S(P_i-1)\Delta\hat{u}(k+1|k)+\cdots+ \\ &\quad S(P_i-H_u+1)\Delta\hat{u}(k+H_u-1|k)\end{aligned} \tag{22}$$

which verifies (1.24).

1.5 The reference trajectory values at the two coincidence points are the same whichever model is used, so we calculate these first. Let t denote the current time. The initial error is $\epsilon(t)=3-1=2$, so the reference trajectory 6 sec ahead is $r(t+6|t)=3-\exp(-6/5)\times 2=2.3976$, and 10 sec ahead is $r(t+10|t)=3-\exp(-10/5)\times 2=2.7293$. Thus the target vector needed in (1.22) is

$$\mathcal{T}=\begin{bmatrix}2.3976\\2.7293\end{bmatrix} \tag{23}$$

Since the output is constant at 1, the input must be constant. The steady-state gain of the model is 2, so this constant value of the input must be 0.5. The free response, with $u(t+\tau)=0.5$ for $\tau>0$, is $y_f(t+\tau)=1$, since the output would remain at its equilibrium value if the input value did not change.

(a). The only information needed from the continuous-time model is the step response values at the coincidence points. The transfer function corresponds to the differential equation

$$y(t) + 7\dot{y}(t) = 2u(t - 1.5) \tag{24}$$

so the step response is the solution to

$$y_f(t) + 7\dot{y}_f(t) = \begin{cases} 0 & (t < 1.5) \\ 2 & (t \geq 1.5) \end{cases} \tag{25}$$

with initial condition $y(0) = 0$, which is $y(t) = 2(1 - e^{-(t-1.5)/7})$ for $t \geq 1.5$ (from the elementary theory of differential equations). Hence the vector of step responses at the coincidence points is

$$\mathcal{S} = \begin{bmatrix} 0.9484 \\ 1.4062 \end{bmatrix} \tag{26}$$

(This can also be obtained by using the function `step` in *MATLAB*'s *Control System Toolbox.*) So now we can apply (1.22) to get

$$\Delta u(\hat{k|k}) = \mathcal{S} \backslash (\mathcal{T} - \mathcal{Y}_f) = 1.3060 \tag{27}$$

so the optimal input is

$$u(k) = 0.5 + 1.3060 = 1.8060 \tag{28}$$

(b). An equivalent discrete-time model is obtained most easily using *MATLAB*'s *Control System Toolbox* function `c2d` on the original transfer function without the delay:

```
sysc=tf(2,[7,1])
sysd=c2d(sysc,0.5)
```

which gives the z-transform transfer function $0.1379/(z - 0.9311)$. Now the delay of 1.5 sec, namely three sample periods, can be incorporated by multiplying by z^{-3}: $0.1379/z^3(z - 0.9311)$. Now using the function `step` gives the step response at the coincidence points as 0.9487 and 1.4068, respectively. Proceeding as in (a), we get $\Delta u(\hat{k|k}) = 1.3055$ and hence $u(k) = 1.8055$.

The two points made by this exercise are: (1) Continuous-time models can be used directly. (2) Whether a continuous-time or a discrete-time model is used makes little difference, if the sampling interval is sufficiently small.

1.6 Since the plant has a delay of 1.5 sec, the predicted output is not affected by the input within that time. So choosing a coincidence point nearer than 1.5 sec into the future would have no effect on the solution.

1.7 With only one coincidence point (assumed to be P steps ahead) and $H_u = 1$ we have $\mathcal{T} = r(k+P|k)$, $\mathcal{Y}_f = y_f(k+P|k)$, and $\Theta = S(P)$, so (1.31) becomes

$$\Delta\hat{u}(k|k) = \frac{r(k+P|k) - d(k) - y_f(k+P|k)}{S(P)} \tag{29}$$

In steady state (1.33) and (1.34) become

$$\begin{aligned} \Delta\hat{u}(k|k) &= \frac{r_P - d_\infty - y_{fP}}{S(P)} && (30) \\ &= \frac{r_P - y_{p\infty} + y_{m\infty} - y_{fP}}{S(P)} && (31) \end{aligned}$$

(1.35) becomes $y_{m\infty} - y_{fP} = 0$, and (1.36) becomes

$$\Delta\hat{u}(k|k) = \frac{r_P - y_{p\infty}}{S(P)} \tag{32}$$

But in the steady state $\Delta\hat{u}(k|k) = 0$ and hence $r_P = y_{p\infty}$. But $r_P = s_\infty - \lambda^P(s_\infty - y_{p\infty}) = s_\infty - \lambda^P(s_\infty - r_P)$. Hence $y_{p\infty} = r_P = s_\infty$.

1.8 To do this exercise, simply change files `basicmpc.m` and `trackmpc.m` as follows. First define the plant by replacing the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW PLANT %%%%%
nump=1;
denp=[1,-1.4,0.45];
plant=tf(nump,denp,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW PLANT %%%%%
```

by the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW PLANT %%%%%
nump=1;
denp=[1,-1.5,0.5];    % (z-0.5)(z-1)
plant=tf(nump,denp,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW PLANT %%%%%
```

then define the model by replacing the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW MODEL %%%%%
model = plant;
%%%%% CHANGE UP TO HERE TO DEFINE NEW MODEL %%%%%
```

by the lines:

```
%%%%% CHANGE FROM HERE TO DEFINE NEW MODEL %%%%%
numm=1.1;
denm=[1,-1.5,0.5];
model = tf(numm,denm,Ts);
%%%%% CHANGE UP TO HERE TO DEFINE NEW MODEL %%%%%
```

1.9 With the independent model implementation, the measured plant output does not directly influence the model output. When $T_{ref} = 0$ the reference trajectory does not depend on the measured plant output, because it is equal to the future set-point trajectory. (Whereas with $T_{ref} \neq 0$ it does depend on the measured output.) Thus for each coincidence point the controller is concerned only to move the output from the present model output value $y_m(k)$ to the required value $s(k + P_i)$. Neither of these values, nor the free response of the model, is affected by the measurement noise. Hence the control signal is not affected by the noise, either.

When offset-free tracking is provided, the disturbance estimate $d(k) = y_p(k) - \hat{y}(k|k-1)$ is affected by the noise, through the measurement $y_p(k)$, and this directly affects the control signal.

Comment: This shows how essential it is to tie the model output to the plant output in some way. Otherwise the model and plant could drift arbitrarily far apart. The way it is done in Section 1.5 is a very easy way of doing it, but not the only possible one.

1.10 This exercise can be solved just by editing the files `basicmpc.m` and `trackmpc.m`, to define the variables `plant`, `model`, `Tref`, `Ts`, `P` and `M` appropriately.

1.11 Just edit the file `unstampc.m` in the obvious way — change the definition of variable `model`.

1.12 The text following equation (1.31) shows what has to be done: in file `unstampc.m` replace the lines:

```
% Compute input signal uu(k):
  if k>1,
    dutraj = theta\(reftraj-ymfree(P)');
    uu(k) = dutraj(1) + uu(k-1);
  else
    dutraj = theta\(reftraj-ymfree(P)');
    uu(k) = dutraj(1) + umpast(1);
  end
```

by:

```
% Compute input signal uu(k):
  d = yp(k) - ym(k);
  if k>1,
    dutraj = theta\(reftraj-d-ymfree(P)');
    uu(k) = dutraj(1) + uu(k-1);
  else
    dutraj = theta\(reftraj-d-ymfree(P)');
    uu(k) = dutraj(1) + umpast(1);
  end
```

1.13 In the existing file `unstampc.m` the restriction that the plant and model should have the same order arises in the lines

```
% Simulate model:
  % Update past model inputs:
  umpast = [uu(k);umpast(1:length(umpast)-1)];
  % Simulation:
  ym(k+1) = -denm(2:ndenm+1)*ympast+numm(2:nnumm+1)*umpast;
  % Update past model outputs:
  ympast = yppast; %%% RE-ALIGNED MODEL
```

In particular the product `denm(2:ndenm+1)*ympast` assumes that the length of the vector `denm(2:ndenm+1)`, which depends on the model order, is the same as the length of the vector `yppast` (since `ympast` is set equal to `yppast`), which depends on the order of the plant.

To handle different orders of plant and model it is necessary to store the appropriate number of past values of the plant output in the vector `ympast`. If the model order is smaller than the plant order this is very easy: just truncate the vector `yppast` (since the most recent plant output values are stored in the initial locations of `yppast`):

```
% Simulate model:
  % Update past model inputs:
  umpast = [uu(k);umpast(1:length(umpast)-1)];
  % Simulation:
  ym(k+1) = -denm(2:ndenm+1)*ympast+numm(2:nnumm+1)*umpast;
  % Update past model outputs:
  if ndenm <= ndenp,           %%% MODEL ORDER <= PLANT ORDER
    ympast = yppast(1:ndenm); %%% RE-ALIGNED MODEL
  end
```

If the model order is larger than the plant order then there are not enough past values in `yppast`, as currently defined. One solution is to leave `yppast` unchanged, but to hold the additional past values in `ympast`, by 'shuffling' them up to the tail of the vector:

```
if ndenm > ndenp,           %%% MODEL ORDER > PLANT ORDER
  ympast(ndenp+1:ndenm) = ympast(ndenp:ndenm-1);
  ympast(1:ndenp) = yppast;
end
```

1.14 The vector $\Delta\mathcal{U}$ should contain only the changes of the control signal:

$$\Delta\mathcal{U} = \begin{bmatrix} \Delta\hat{u}(k+M_1|k) \\ \Delta\hat{u}(k+M_2|k) \\ \vdots \\ \Delta\hat{u}(k+M_{H_u}|k) \end{bmatrix} \tag{33}$$

The matrix Θ should contain the columns corresponding to these elements of $\Delta\mathcal{U}$, so that (1.27) is replaced by:

$$\Theta = \begin{bmatrix} S(P_1-M_1) & S(P_1-M_2) & \cdots & S(P_1-M_{H_u}) \\ S(P_2-M_1) & S(P_2-M_2) & \cdots & S(P_2-M_{H_u}) \\ \vdots & \vdots & \vdots & \vdots \\ S(P_c-M_1) & S(P_c-M_2) & \cdots & S(P_c-M_{H_u}) \end{bmatrix} \tag{34}$$

(remember that $S(i) = 0$ if $i < 0$). Note that the vector $\mathcal{Y}$ remains the same as in (1.26).

1.15 Exercise 1.15 is done by editing the files `basicmpc`, `trackmpc` and `noisympc`. The main point to be made here is that the effects of changing various parameters are difficult to predict precisely, and can occasionally be counter-intuitive. This provides some motivation for Chapter 7, which discusses tuning MPC controllers.

Obvious things to check are:

(a). Increasing T_{ref} slows down the response.

(b). Increasing the prediction horizon (the location of the last coincidence point) improves closed-loop stability (because the controller pays more attention to long-term consequences of its actions).

(c). Increasing H_u increases the computation time. (Use *MATLAB* functions `tic` and `toc` to check this.)

Chapter 2

A basic formulation of predictive control

2.1 The following *MATLAB* code does the job:

```
Q = [10 4 ; 4 3]
eig(Q)
```

This shows the eigenvalues are 11.815 and 1.185, which are both positive.

2.2 (a).

$$V(x) = 9x_1^2 + 25x_2^2 + 16x_3^2 = [x_1, x_2, x_3]\begin{bmatrix} 9 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 16 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1}$$

so $Q = \text{diag}[9, 25, 16]$.

(b). With $x^T = [x_1, x_2, x_3]$ and $u^T = [u_1, u_2]$ we get $V(x,u) = x^TQx + u^TRu$ if

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 100 & 0 \\ 0 & 4 \end{bmatrix} \tag{2}$$

(c). Yes. If $x_i \neq 0$ for any i then both $V(x)$ in (a) and $V(x,u)$ in (b) are positive, by inspection. Similarly, if $u_i \neq 0$ for any i then $V(x,u)$ in (b) is positive. If $x = 0$ then $V(x) = 0$. If $x = 0$ and $u = 0$ then $V(x,u) = 0$. So both functions V are positive-definite.

2.3 *Error in question:* The formula for the gradient is not (2.19), but the formula for ∇V which appears at the end of Mini-Tutorial 1.